Two-Dimensional Parallel Solver for the Solution of Navier-Stokes Equations with Constant and Variable Coefficients using ADI on Cells

appeared in Parallel Computing, 24, pp. 673-699, 1998.

A. Averbuch L. Vozovoi
School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel
L. Ioffe M. Israeli*
Faculty of Computer Science, Technion, Haifa 32000, Israel

Abstract

The paper proposed a new algorithm for the parallel solution of two-dimensional Navier-Stokes type equation with constant and non-constant coefficients which is mapped onto cell topology. This paper is a further development in the application of the local Fourier methods to the solutions of PDE's in multidomain regions.

The extension of the above solution to problems with non-constant coefficients is suggested via spectral multidomain preconditioner. This approach is efficient when we have good local approximations in each subdomain. By dividing the computational domain into a large enough number of subdomains we can guarantee it.

The new achievement here is that we are able to handle decomposition of the domain into cells that is the decomposed in both directions, x and y.

An appropriate Alternate Direction Implicit (ADI) scheme was developed. It enables the reduction of a 2-D problem to a collection of uncoupled 1-D ODE's. In effect, the 1-D solver becomes the basic routine to solve a 2-D problem using splitting of the differential operators by ADI.

Detailed performance analysis is given where the issue of the communication among the domains (processors) is examined. We show that by using the Richardson method only local communication is required.

The algorithm was implemented on IBM SP2, network of Alpha workstations, and MOSIX[9, 10] which is a network of i586. All are implemented using the PVM software package and the same ADI program was running on these different multiprocessor configurations. It achieved efficiency of 55-70% depending on the multiprocessor.

Key-Words: Parallel implementation, domain decomposition, matching, ADI, spectral preconditioner, local Fourier basis.

*M. Israeli was supported by the fund for the promotion of research at the Technion

1 Introduction

The high performance of parallel machines can be realized only if effective parallelizing algorithms are supplied.

The Multidomain Local Fourier Method (MDLF) approach[1]-[6], based on the local Fourier methods[7], overcomes most of the global coupling, inherent both in the use of a spectral method in space and in the implicit discretization of time in parallel processing.

In [1] and [6] we presented parallel multidomain algorithms for the solution of the incompressible Navier-Stokes equations. A high-order scheme is employed for the time discretization. The discretization in space is based on Fourier methods. For the interior subdomains we use an overlapping Local Fourier Basis technique, and for the boundary subdomains - a non-overlapping Fourier-Gegenbauer method [16].

The matching of the local solutions is performed via a direct pointwise procedure on the interfaces, using properly weighted Green's functions on the interface. The unknown coefficients are found explicitly in terms of the jumps of the solution and its first derivatives on the interfaces. The localization properties on the interface of the Green's functions are exploited in order to simplify the matching relations so that communication is reduced mostly to local data exchanges between neighboring subdomains (processors).

In [1, 6] a low communication multidomain approach was implemented. A notable feature of this implementation is that it makes use of Fourier methods for space discretization. A modified Local Fourier Basis (LFB) technique [2]-[4] is employed for the smooth decomposition of the original problem into subproblems. The use of the Fourier basis leads to a great simplification of the matching problem as it makes it possible to match independently each separate harmonic in the spectral space. Another important feature of this approach is that it takes advantage of the local behaviour of the Green's functions which is used to impose continuity conditions on the interfaces. As a result, the influence of remote domains on the processing at a particular location become negligible and the interprocessor communication is confined to local data exchange between neighboring processors.

As described in sections 2.2 and 2.3 we use two approaches in order to preserve spectral accuracy: a modified Local Fourier Basis (LFB) method [7], implemented in the interior subdomains where a smooth decomposition of the source function is performed by using a system of overlapping bell functions, and the Fourier-Gegenbauer (FG) method [16] which makes use of the re-expansion of the (local) Fourier partial sums into rapidly convergent Gegenbauer series. The FG technique is implemented in the subdomains adjacent to the boundaries where overlapping of two contiguous subdomains is not possible in the case of non-periodicity (e.g. for Dirichlet or Neumann boundary conditions). The FG method is a good choice for our purpose because it operates inside the interval where the function is defined, so that it does not require overlapping of subdomains. Also, the FG method has uniform resolution like the LFB method, so that both Fourier techniques are easily combined. Thus, the use of a non-overlapping FG method enables us to treat non-periodic problems.

When we use the Poisson equation[6], which describes equilibrium processes, we have global interactions. Therefore, the solution of this equation requires global communication between subdomains. Nevertheless, even in this case we demonstrated in [6] that in two (or more) dimensions the necessary global communication constitutes of only a small percentage of the required communication.

This paper is a further development of the local Fourier methods for the solution of PDE's in multidomain regions. The paper suggests a new algorithm for the parallel solution of two-dimensional equation of the Navier-Stokes type [15] with constant and non-constant which is mapped onto cell topology.

The extension of the above solution to problems with non-constant coefficients is suggested via spectral multidomain preconditioner. This approach is efficient when we have good local approximations in each subdomain (small variations in the coefficients from their mean values). By dividing the computational domain into a large enough number of subdomains we can guarantee it. This is described in section 3.3.

An appropriate ADI scheme was developed. It enables the reduction of a 2-D problem to a collection of uncoupled 1-D ODE's. In effect, the 1-D solver becomes the basic routine to solve a 2-D problem using splitting of differential operators by ADI. The ADI is only of second order accuracy when there exists at least one periodic direction, and it may be even of lower order when there is no such direction. In addition, it has a perceivable splitting error. But for our purpose it is indeed an appropriate method to utilize.

Detailed performance analysis is given where the issue of the communication among the domains (processors) is examined. This is of particular importance when using iterative methods. We show that by using the Richardson method only local communication is required. On the other hand, this method converges slower than Conjugate Gradient (CG) type methods using "global" computation (scalar products). Therefore, there is a tradeoff between these approaches: "local" (Richardson method-slow convergence) and "global" (CG method-fast convergence).

The 2-D Navier-Stokes solver was implemented on three different MIMD message-passing multiprocessors: a 60-processor IBM SP2, a 20-processors MOSIX[9, 10], and a network of 10 Alpha workstations using the PVM (parallel virtual machine) software package[8]. The same program without any modification was run on all these computational platforms, which demonstrates the portability of the algorithms and its software. It achieved efficiency of 55-70% depending on the multiprocessor.

The proposed algorithm has the potential for treating efficiently complex geometries. This is a work in progress.

The paper has the following structure: in section 2.1 we describe the local spectral Fourier method that we use for the solution and its application to 2-D cell topology. In section 3 we describe the ADI method where in section 3.2 we describe the parallel algorithm for the cell computation with constant coefficients. In section 3.3 the algorithm for the solution with non-constant coefficients that is based on spectral preconditioner (section 3.3.1). In section 3.4 we describe the parallel algorithm for the cell computation with non-constant coefficients. The model problem is described in section 4. In section 5 we describe the detailed performance analysis of the model problem implemented on SP2, MOSIX and network of ALPHA workstation.

2 Multidomain Local Fourier Method (MDLF)

2.1 Introduction

Domain decomposition techniques have become an efficient tool for parallelizing algorithms, particularly, due to its ability to divide the computational work among different processors and then reconstruct a global solution from the solutions in each domain. Domain decomposition can be combined successfully with spectral methods using series expansions into orthogonal bases. These methods are particularly suitable for computation of the incompressible Navier-Stokes equations with high Reynolds numbers, since they converge exponentially fast as the number of degrees of freedom N increases. Polynomial (Chebyshev or Legendre) bases are mostly used for problems that were spread into multidomains. Spectral methods with the Fourier basis are inefficient for the construction of *non-periodic* local solutions in subdomains. The reason is that the truncated Fourier series of a non-periodic function (having a discontinuous periodic extension at the boundaries) converges very slowly, like $1/N$, inside the region, and has $O(1)$ spurious oscillations near the boundaries (Gibbs phenomenon).

However, the Gibbs phenomenon can be avoided if a smooth decomposition of functions is performed. A spectrally accurate multidomain Fourier method was developed in [2]. This method employs a particular version of the projection technique introduced in [7] where the construction of the Local Fourier Basis (LFB) was introduced. The multidomain local Fourier method (MDLF) of [2]-[5] is especially efficient for large problems using high resolutions in space. In this case, the number of operations required to perform the projection is insignificant in comparison with what is needed to execute the Fast Fourier Transform (FFT).

An important advantage of this method, when compared to other multidomain spectral techniques, is that it enables a great simplification of the matching relations for the interface unknowns. The matching of the local solutions constructed independently in each subdomain is necessary to obtain a continuous global solution. For problems with constant coefficients and in simple domains, the use of the Fourier basis enables us to fulfill the matching of each harmonic separately, and thus to eliminate the global coupling of the interface unknowns.

The matching relations can be simplified furthermore by utilizing localization properties of an elliptic operator resulting from the discretization in time of a time-dependent problem [3]. Eventually, only *local communication* between neighboring is necessary. Thus, the MDLF algorithm is completely scalable in many cases and possesses high parallel efficiency [1, 6].

The ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem indicates that we have *scalable* parallel systems. The *scalability* of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to utilize increasing processing resources effectively.

Prior to the proposed algorithm when we decompose into strips[1, 6], we are able to "kill" large variances in the coefficients only in the direction across strips while along the strips the variance was out of control. The new achievement here is that we are able to handle decomposition into cells where decomposition is done in both directions while preserving small variances on each subdomain.

2.2 Application of the MDLF Method to Strip Topology

In this section we discuss briefly the 1-D MDLF method and its extension to 2-D for the simplest rectangular domains decomposed into parallel strips while parallel cells is discussed in the next section and it is based on [5]. A detailed description of this method is given in [2]-[5].

We describe the numerical algorithm as applied to the boundary value problem of the diffusion type

$$u'' - \lambda^2 u = f(x), \qquad x \in [0, L], \tag{2.1}$$

with periodic boundary conditions

$$u(0) = u(L), \qquad u'(0) = u'(L). \tag{2.2}$$

We divide the computational interval $[0, L]$ into P pieces (subdomains) of an arbitrary size and discretize the local subproblems on a uniform grid of collocation points.

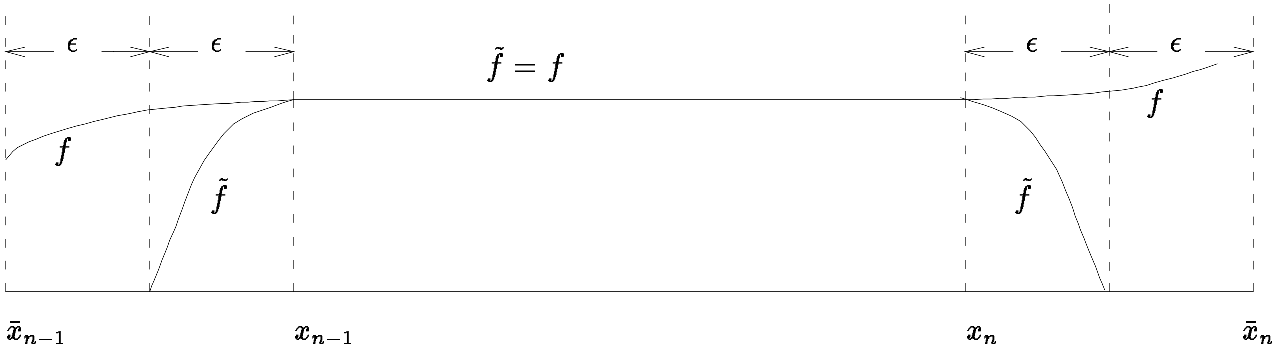

Figure 1: The folding operation

Following [7], we cover the interval with the collection of smooth overlapping bell functions $b_n(x)$, which satisfy certain properties to allow the projection in L_2. In our particular version of the LFB method the bell functions b_n is 1 inside the subdomains $l_n = [x_{n-1},\ x_n]$ and smoothly decay outwards over the intervals 2ϵ (see Figs. 1 and 2). The algorithm consists of two steps:

I. Construction of the local solutions in each subdomain -

For each subdomain we perform:

1. Extension of the local source function $f^{(n)}(x)$ beyond the interfaces over the distance 2ϵ (from each side). This is done by exchanging data with two neighboring subdomains (see Fig. 2). For periodic problems, as in Eq. (2.2), the continuous extension beyond the boundaries, $x < 0$ and $x > L$, can be obtained by overlapping the first and the last subdomains (cyclic boundary conditions).

2. Projection of $f^{(n)}(x)$ using the bell functions b_n. The projection procedure results in a smooth function $\tilde{f}^{(n)}(x)$ localized on the extended interval $\bar{l}_n = [\bar{x}_{n-1} - 2\epsilon, \bar{x}_n + 2\epsilon]$ which can be represented by a rapidly convergent Fourier series. This step results in a periodic continuous functions $f^{(n)}(x)$. Due to the choice of b_n (1 inside the interval), this function coincides with the original one, $f^{(n)}(x)$, on the range l_n of the subdomain.

3. Solution of the elemental subproblems (Eq. (2.1)) with the smoothed source functions $\tilde{f}^{(n)}$ in the RHS by means of the pseudospectral technique. It consists of the integration in the spectral space and two Fast Fourier Transforms (FFT) from the physical to the spectral space and vice versa. We apply the smoothing algorithm to the pieces of the source function $f^{(n)}(x)$. Now, equation (2.1), with the function $\tilde{f}$ in the RHS, can be solved with spectral accuracy.

   The smoothing procedure does not change the primary source function *inside* the subdomains $[\bar{x}_{n-1}, \bar{x}]$ but only distorts the extra pieces on the *external* intervals $[\bar{x}_{n-1} - \epsilon, \bar{x}_{n-1}]$ and $[\bar{x}_n, \bar{x} + \epsilon]$. Therefore, within each subdomain the solution $\tilde{u}^{(n)}(x)$ of the "smoothed" problem obviously coincides with the particular solution $u_p^{(n)}(x)$ of the original problem (2.1). Hence, our approach does not require any "unfolding" operations to recover the original non-periodic solution. (The unfolding is the inverse operation associated with the LFB construction which recovers the original function from its folding operation). The only extra work is the overlapping which have to be done in order to replace the solution of the non-periodic local problems in subdomains by the solution of the smooth periodic problems using the spectral Fourier method.

   The computational expenses for these external intervals are considered as extra work in order to avoid the Gibbs phenomenon when the Fourier method is applied to non-periodic problems. It can be shown that for large problems using high resolution in space this extra work is a small percentage of the whole computational work.

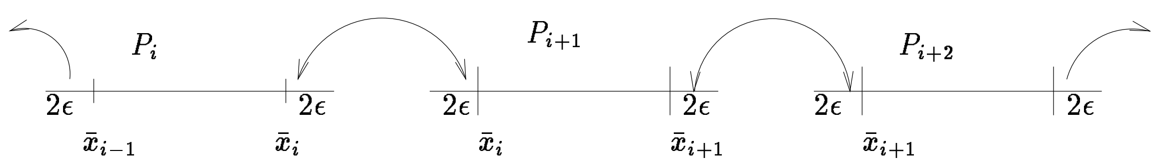

Figure 2: Transfer of overlapped contiguous subdomains between neighboring processors

II. Matching of the solutions from each subdomain -

The global solution $u_p = \bigcup_{n=1}^{P} u_p^{(n)}$ will be piecewise continuous since the local solutions usually have jumps at the interfaces.

In order to obtain the continuous global solution, we correct the particular solutions, constructed at the previous step, by proper weighted local homogeneous solutions of

Eq. (2.1):

$$u = \bigcup_{n=1}^{P} u^{(n)}, \quad u^{(n)} = u_p^{(n)} + A_n h_+^{(n)} + B_n h_-^{(n)}. \tag{2.3}$$

These solutions are based on two exponential functions $h_-^{(n)} = e^{-\lambda x}$, $h_+^{(n)} = e^{-\lambda(l_n - x)}$, $0 < x < l_n$, which decay inward the subdomain. The coefficients A_n, B_n have to be found in order to enforce the continuity of $u(x)$ and $u'(x)$ on the interfaces. They can be expressed, in closed form, in terms of jumps of particular solutions u_p and u_p' at the interfaces.

It seems that computation of matching coefficients usually requires global communication among the processors since all the interfaces contribute to the matching relations. This can reduce (degrade) the achieved parallel speedup. We observe, however, that if the parameter λ in Eq. (2.1) is much greater than 1, then the functions $h_\pm(x)$ decay rapidly away from the interfaces. Then, for each particular location x, the influence of the remote interfaces is negligible. As a result, the matching relations decouple in such a way that the computation of each pair of the matching parameters A_n, B_n requires only the knowledge of jumps at two nearest interfaces which implies that only local communication is involved.

To conclude, the computation of the particular local solutions and the matching procedure use only local, neighbor-to-neighbor, inter-processor communication that makes the whole algorithm as an ideal candidate to become a scalable algorithm.

The present algorithm can be extended to solve non-periodic problems. For such problems the continuous extension of functions beyond the boundaries is not available. Therefore, the boundary subdomains should be treated differently. An alternative approach consists of re-expanding the Fourier partial sums, with its Gibbs phenomenon, into a rapidly converging Gegenbauer series as was done in [16].

The extension of this method to rectangular domains decomposed into parallel strips, is straightforward. After applying the FFT along the strips, we arrive at a collection of *uncoupled* 1-D ODEs for the Fourier coefficients. These equations are of the same type as Eq. (2.1). Therefore, they can be solved by using the 1-D MDLF routine. The extension of this method to cell topology is described in section 2.3.

2.3 Application of the MDLF Method to Cell Topology

We solve the two-dimensional Helmholtz equation

$$\nabla^2 u - \lambda^2 u = f(x, y). \tag{2.4}$$

Suppose that the source function $f(x, y)$ is periodic in the y direction and we apply the FFT along the strips. Then, the problem is reduced to a set of one-dimensional problems for the Fourier coefficients $\hat{u}_k^{(n)}(x)$:

$$\hat{u}_k^{(n)''} - (k^2 + \lambda^2)\hat{u}_k^{(n)} = \hat{f}_k^{(n)}(x) \qquad \text{in } \omega_n, \qquad k = -\frac{N_y}{2}, \ldots, \frac{N_y}{2} \tag{2.5}$$

(N_y is the number of collocation points in the y direction), which can be solved by using the algorithm with the local matching on the interfaces (as described in section 2.2).

If the function $f(x,y)$ is not periodic along the strips, as it was assumed before, it can be transformed to a periodic form by using the same "folding" procedure (described in section 2.2). Then, the previous situation is reproduced on an extended region $\bar{\Omega} = (0, L_x) \times (-\epsilon, L_y + \epsilon)$. The solution of this modified problem will coincide with the original one on Ω.

Assume that our problem is mapped into a domain which is decomposed into cells. We showed in [5] that the implementation of the LFB technique enables us to reduce the overall coupling of the interface points to a sequence of simple, one dimensional local matching relations. The idea is to apply this algorithm in the x and y directions separately.

In the first step we take a horizontal row of cells and transform the source function using the LFB in the y-direction. (The procedure includes an ϵ-extension and smoothing across the horizontal interfaces, anti-symmetrical reflection and application of FFT on the local period).

In the second step we solve the system of one-dimensional problems of type (2.5) for the Fourier coefficients with the local matching on the vertical interfaces and then transform back into the physical space. The resulting particular solution $u^{(I)}$ will be smooth in each horizontal band except for the horizontal interfaces.

To remove discontinuities from the horizontal interfaces we transform the function $u^{(I)}(x,y)$ using the LFB in the x-direction in each vertical column and match Fourier coefficients on these lines. Then, after the application of the inverse transform into the physical space, we get a vertically smooth solution $u^{(II)}$.

Smoothing in the $y-$direction introduces discontinuities on the vertical interfaces because each Fourier coefficient is matched independently. These jumps can be removed by another matching (in the $x-$direction). Then, this is done in each new cycle along x and y alternatively. This results in a smoother solution.

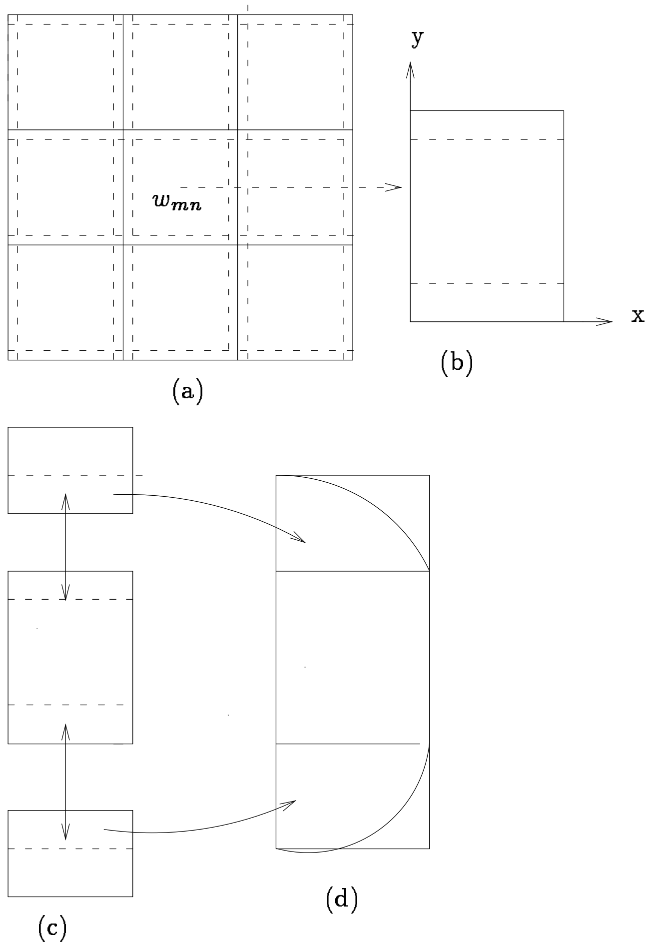

Figure 3: Implementation of the cell algorithm: (a) The domain is divided into $P_x \times P_y$ processors. (b) The overlapped area of each cell in one direction marked by the dotted lines. (c) and (d) show the extension and smoothing of the local solution.

We solve the two-dimensional Helmholtz equation (2.4) in the rectangular cells $\Omega = \bigcup_{n=1}^{P_x} \bigcup_{m=1}^{P_y} \omega_{mn}$ (Fig. 3(a)) of sizes $l_x \times l_y$; $l_x = L_x/P_x$, $l_y = L_y/P_y$ (Fig. 3(a)). P_x and P_y are the number of processors along the x and y axis respectively.

Instead of *simultaneous* matching at all collocation points we rather match *successively* in x and y directions.

1. We take horizontal row of cells $\Omega_\nu = \bigcup_{m=1}^{P_x} \omega_{\nu m}, \nu = 1, \ldots, P_y$ and transform the source function $f(x, y)$ into the LFB in the y-direction (the procedure includes an ϵ-extension, smoothing across the boundaries $\bar{y} = (\nu - 1)l_y - \epsilon$ and $\bar{y} = \nu l_y + \epsilon$, antisymmetric reflection and FFT on the local period $\tau = 2(l_y + 2\epsilon)$- see section 2.2 and Figs. 3 (c) and (d)).

2. We solve the system of one-dimensional problems of the type

$$\frac{d^2 \hat{U}_k^{(n)}}{dx^2} - (k^2 + \lambda^2)\hat{U}_k^{(n)} = \hat{f}_k^{(n)}(x) \qquad \text{in } \omega_n, \qquad k = -\frac{N_y}{2}, \ldots, \frac{N_y}{2}$$

(N_y is the number of collocation points in the y direction) for the Fourier coefficients with the local matching on the vertical interfaces, after which we transform back into the physical space.

3. The resulting particular solution $U_p(x, y)$ will be smooth in each horizontal band except for the interfaces $\bar{y}_n = nl_y,\ n = 1, \ldots, P_x$ (figure 3(c,d)).

4. To remove discontinuities from the horizontal interfaces we use again our one-dimensional matching routine. We take the vertical column $\Omega^\mu = \bigcup_{n=1}^{P_y} \omega_{n\mu},\ \mu = 1, \ldots, P_x$, transform U_p into the LFB (by analogy with the source function on the first step) and implement the matching on the lines $\bar{y}_n$ for each Fourier coefficient. By inverting the transform into the physical space we get the vertically smooth solution ${U^I}_p$.

   Smoothing in the y direction induces discontinuities on the vertical lines $\bar{x}_n = nl_x,\ n = 1, \ldots, P_x$, because each Fourier coefficient is matched independently. These jumps are removed by another matching (in the x direction). Each new "ironing" along x and y alternatively results in a smoother solution.

The proposed algorithm is an extension (generalization) of the strip[1, 6] computational algorithm since it can be done in the following way:

1. Do steps (a)-(d) as described in Fig. 3. In other words, as in the LFB, we take extension and perform smoothing of the local particular solution U_p.

2. As in the strip computation, we match in the x direction.

3. Extension, smoothing and FFT as explained in the strip computation.

4. As in the strip computation, we match in the y direction.

5. Go to step 2.

3 The ADI Method

The alternate direction implicit (ADI) method is another time-discretization scheme. In this section we describe how the ADI method was implemented for the domain decomposition in the form of cells. The performance analysis of this implementation related to cells is described in sections 3.2 and 3.3.

3.1 Discretization in Time

An efficient time-discretization scheme can be constructed by using the alternate direction implicit (ADI) method. This is a two-step method which can be used as

$$\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t/2} = \nu(\mathbf{v}^*_{xx} + \mathbf{v}^n_{yy}) - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.1}$$

$$\frac{\mathbf{v}^{n+1}-\mathbf{v}^*}{\Delta t/2} = \nu(\mathbf{v}^*_{xx}+\mathbf{v}^{n+1}_{yy}) - \sum_{q=0}^{J_e-1}\beta_q\left(\mathbf{N}(\mathbf{v}^{n-q})-\mathbf{F}^{n-q}\right) \tag{3.2}$$

When the $2-d$ order Adams-Bashford scheme is employed for the approximation of the non-linear and the non-homogeneous terms. The following parameters are used:
$J_e = 2,\ \beta_0 = 1.5,\ \beta_1 = 0.5.$

The ADI method allows us to split a 2-D problem into two *ordinary* 1-D equations such that the first one (3.1) is solved in the x direction, and the second equation (3.2) is solved in the y direction. The main advantage of this approach is that it enables to perform the time-advancing step in the *physical* space without the need initially to apply the transform into the spectral space in the transverse direction (as was done in the previous sections and in [1]).

We rewrite the system (3.1)-(3.2) in the form

$$\mathbf{v}^*_{xx} - \lambda^2\mathbf{v}^* = -\lambda^2\mathbf{v}^n - \mathbf{v}^n_{yy} + \frac{1}{\nu}\mathbf{G} \equiv \mathcal{R}_1 \tag{3.3}$$

$$\mathbf{v}^{n+1}_{yy} - \lambda^2\mathbf{v}^{n+1} = -\lambda^2\mathbf{v}^* - \mathbf{v}^*_{xx} + \frac{1}{\nu}\mathbf{G} \equiv \mathcal{R}_2 \tag{3.4}$$

where

$$\lambda^2 = \frac{2}{\nu\Delta t},$$

$$\mathbf{G} = \frac{3}{2}(\mathbf{N}-\mathbf{F})^n - \frac{1}{2}(\mathbf{N}-\mathbf{F})^{n-1}$$

The most precise and efficient way to evaluate the non-homogeneous terms in the RHS of (3.3)-(3.4), which does not require any additional calculation of derivatives, is done in the following way:

$$\mathcal{R}_1 = -\mathcal{R}_2 - 2\lambda^2\mathbf{v}^{n+1} + \mathbf{G}/\nu, \tag{3.5}$$

$$\mathcal{R}_2 = -\mathcal{R}_1 - 2\lambda^2\mathbf{v}^* + \mathbf{G}/\nu \tag{3.6}$$

Equation (3.3) is solved for $\mathbf{v}^*(x_i, y_j)$ at each $y_j = j\Delta y,\ \Delta y = 2\pi/N_y,\ j = 0, 1, ..., N_y$ and Eq. (3.4) is solved for $\mathbf{v}^{n+1}$ at each $x_i = i\Delta x,\ \Delta x = 2\pi/N_x,\ i = 0, 1, ..., N_x$ by using the LFB method.

The ADI method can be implemented for domain decomposition in the form of cells as well as strips[1]. In the case of cells, the matching is performed both on the vertical (for Eq. (3.3)) and the horizontal interfaces $\bar{y}_r = r\ l_y,\ \ r = 1, 2, ..., P_y,\ l_y = 2\pi/P_y$ for Eq. (3.4).

Therefore, the general ADI computational algorithm utilizing domain decomposition in the form of cells is identical to one described in section 2.2 and in [1] but the time-advancing step is performed in physical space without transforming it into spectral space.

3.2 Parallel Algorithm for the Cell Computation with Constant Coefficients

- Initial computation in each processor (no time marching yet)

-Initial

- Computation of the exact solution (for verification of the result)
- Computation of the forcing term according to (Eq. (3.3)) which corresponds to exact solution
- Computation of $\mathcal{R}_1$ without the non-linear terms $\mathbf{N}(\mathbf{v}^{n-q})$)

-Computation of the non-linear terms

-Adding the non-linear terms

Adding the non-linear terms to $\mathcal{R}_1$ (without any time stepping)

-Start of the main loop (advance in time)

Adams-Bashford scheme:

Adams-Bashford scheme is employed for the approximation of the non-linear term. It is performed for both directions (x and y).

Computation of the RHS:

Computation of the RHS according to Eq. (3.5). It is done for the two directions.

Exchange of the overlapped $\mathcal{R}_1$ in the x direction (communication):

Transfer of the overlapped RHS between the two neighbors

Computation of the partial solution:

Computation of the partial solution from Eq. (3.3).

Matching in the x direction (communication):

As explained in section 2.3.

Computation of real velocity

Computation of $\mathcal{R}_2$:

Computation of $\mathcal{R}_2$ according to Eq. (3.6).

Exchange of overlapped $\mathcal{R}_2$ in the y direction (communication):

Transfer of the overlapped RHS between the two neighbors

Computation of the partial solution:

We compute the partial solution from Eq. (3.4).

Exchange of boundary values (communication)

Matching in the y direction (communication):

As explained in section 2.3.

-Going back to the start of the main loop

Computation of the non-linear term in the next time-step.

3.3 Solution with Non-Constant Coefficients

The following section suggests solution to problems with non-constant coefficients via spectral multidomain preconditioner. This approach is efficient when we have a good local approximations in each subdomain (small variations in the coefficients from their mean values). By dividing the computational domain into a large enough number of subdomains we can guarantee it.

3.3.1 Preconditioned Iterative Method with Spectral Preconditioner

If we have

$$\mathcal{L}u = f \qquad \text{in } \omega \tag{3.7}$$

where

$$\mathcal{L}u = \frac{\partial^2 u}{\partial \eta^2} + a(\xi,\eta)\frac{\partial^2 u}{\partial \xi \partial \eta} + b(\xi,\eta)\frac{\partial^2 u}{\partial \xi^2} + c(\xi,\eta)\frac{\partial u}{\partial \eta} + d(\xi,\eta)u. \tag{3.8}$$

then, spectral approximations to the boundary value problem lead to a full $N \times N$ matrix equations for the N expanded coefficients. The solution of this algebraic system requires $O(N^2)$ storage locations and $O(N^3)$ operations to invert the matrix (using the Gauss elimination or other direct methods). Obviously, for multi-dimensional problems of realistic size (typically $N \sim 10^6$) the direct solution of such a system is very expensive.

The straightforward application of the iterative methods to the solution of big linear algebraic systems is also inefficient. The reason is that the typical number of iterations required to reduce the error by an order of magnitude (it can be characterized by the *condition number* κ) grows with the size N of the system. For the second order operator (3.7), $\kappa \propto N^2$ no matter whatever discretization method is being used.

The convergence of the iterations can be improved substantially, if an appropriate *preconditioner* to the original differential operator is used [14]. In this section we construct an efficient *spectral* preconditioner to $\mathcal{L}$ to obtain a rapidly convergent iterative algorithm.

The idea of preconditioning is to solve the equation

$$H^{-1}\mathcal{L}u = H^{-1}f \tag{3.9}$$

instead of Eq. (3.7). Instead of solving the problem directly we solve it iteratively using a preconditioned iterative method The preconditioned iterations (for the particularly simple Richardson scheme) look as follows:

$$Hu^{n+1} = f - (\mathcal{L} - H)u^n. \tag{3.10}$$

Both Eqs. (3.9) and (3.7) are equivalent in view of the solution but the iterative method may converge much faster for Eq. (3.9) if the preconditioner H approximates $\mathcal{L}$ in the sense that the condition number of the "compound" operator $H^{-1}\mathcal{L}$ is bounded (independent of N).

In [13], a low order (finite-difference or finite-element) approximation to $\mathcal{L}$ is used in the capacity of H. The use of a higher order finite-difference preconditioner improves the rate of convergence (in comparison with a lower order approximations) but it partially offsets by the additional cost to invert a denser matrix (representing the operator which is discretized on a larger stencil). Therefore, high-order finite-difference preconditioners are not useful here.

More sophisticated iterative schemes, such as conjugate gradient types method, can be used as well. If the operator $\mathcal{L}$ is approximated in a spectral basis, upon convergence of iterations ((3.10)) a spectrally accurate solution is obtained. The rate of convergence depends on the choice of the preconditioner H.

If the coefficients in $\mathcal{L}$ vary by small ranges, then the *spectral* preconditioner (Eq. (3.10)) obviously provides a better approximation to $\mathcal{L}$ than any low-order preconditioner. However, if the coefficients vary on the computational region, then Eq. (3.10) is a poor approximation to $\mathcal{L}$.

The situation will be changed if the preconditioned iterative method is combined with the domain decomposition technique. Domain decomposition provides us with a new possibility for using *different* preconditioners in the subdomains, where each one is a good constant-coefficient approximation to $\mathcal{L}$ on a small enough patch of the region. In the local preconditioner H_n the coefficients are computed by averaging over the region of the subdomains ω_n. As a result, the convergence rate of the iterations can be considerably improved.

We propose to construct H which will have the same form as $\mathcal{L}$ but with *constant* coefficients instead of having variable coefficients in $\mathcal{L}$. For instance, the preconditioner of the operator (3.7) should be

$$Hu = \frac{\partial^2 u}{\partial \eta^2} + \bar{a}\frac{\partial^2 u}{\partial \xi \partial \eta} + \bar{b}\frac{\partial^2 u}{\partial \xi^2} + \bar{c}\frac{\partial u}{\partial \eta} + \bar{d}u. \tag{3.11}$$

The coefficients $\bar{a}, \bar{b}, \bar{c}$ and $\bar{d}$ can be obtained, for example, by averaging the corresponding non-constant coefficients over the region ω:

$$\bar{b} = \frac{1}{L}\int_{\omega} J^{-1}(\xi, \eta) d\xi d\eta. \tag{3.12}$$

The operator H, which is defined in the rectangular domain ω, is trivially inverted by the Fourier method.

If the coefficients in $\mathcal{L}$ have small variations then the *spectral* preconditioner (3.11) obviously provides a better approximation to $\mathcal{L}$ than any low-order preconditioner. However, if the coefficients vary substantially on the computational region, then (3.11) is going to be a poor approximation to $\mathcal{L}$.

The situation will change if the preconditioned iteration method is combined with the domain decomposition technique. Domain decomposition provides us with a new possibility to use *different* preconditioners in the subdomains, where each one has a good constant-coefficient approximation to $\mathcal{L}$ on a small enough patch of the region. As a result, the convergence rate of the iterations are improved considerably.

We illustrate this conclusion by the following 1-D example:

$$u'' + a(x)u' - \lambda^2 u = f(x), \qquad x \in [0,1] \quad u(0) = u(1) = 0, \tag{3.13}$$

where $a(x) = 10\ (1 + \tan\ 20(x - 0.5))$. The function $a(x)$ changes rapidly from 0 to 20 on a small interval $\Delta x \sim 1/20$ near the point $x = 0.5$. The forcing function has the form $f(x) = 2 - a + (2a + \lambda^2)x - \lambda^2 x^2$ which corresponds to the exact solution $u_{ex}(x) = x(x-1)$.

Following our suggestion, we divide the computational interval $x \in [0,1]$ into a number of subintervals $[x_{n-1}, x_n]$, $n = 1, ..., P$ and define a set of constant coefficient operators H_n as follows:

$$H_n = u'' + \bar{a}_n u' - \lambda^2 u, \qquad x \in [x_{n-1}, x_n], \tag{3.14}$$

where

$$\bar{a}_n = \frac{1}{x_n - x_{n-1}} \int_{x_{n-1}}^{x_n} a(x) dx.$$

Figure 4 shows the variance of the coefficient $a(x)$ on the interval $x \in [0,1]$ and its step-wise approximation $\bar{a}_n$ (appears as a_n in the figure) for $P = 15$.

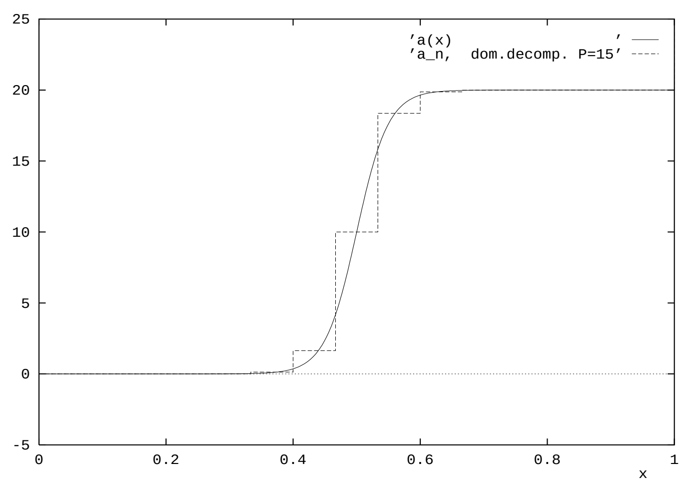

Figure 4: Step-wise approximation of the variable coefficient $a(x) = 10\ (1 + \tan\ 20(x - 0.5))$.

The convergence history of the preconditioned conjugate gradient iterations with the spectral preconditioner (3.14) is given in Fig. 5 for several domain decompositions forms (parameter $\lambda = 1$). When the number of subdomains varies from $P = 11$ to $P = 15$, the same asymptotic accuracy is eventually attained. However, the convergence is much *faster* for larger P, i.e., for smaller deviations $|a(x) - a_n|$ within the intervals $[x_{n-1}, x_n]$. If the deviations $|a(x) - a_n|$ are too large ($P = 9$), then a *smaller* accuracy is achieved after an affordable number (tens) of iterations.

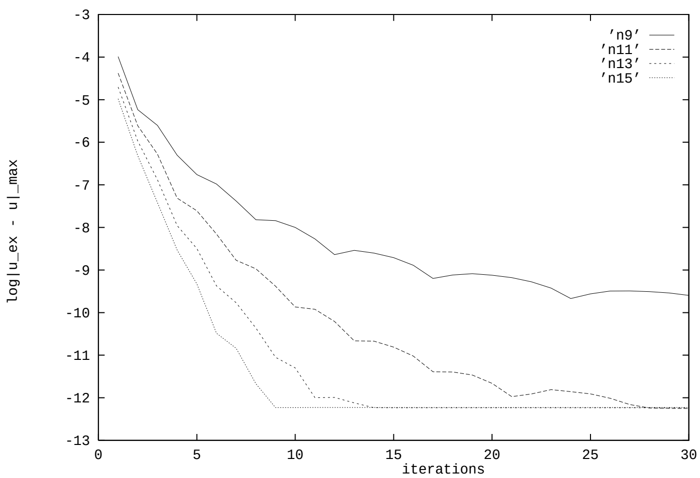

Figure 5: Acceleration of convergence by splitting the computational region into an increasing number of subdomains P. n9,n11,n13,n15 correspond to $p = 9, 11, 13, 15$, respectively.

Note that the computation of inner products, needed when conjugate-gradient type methods is being used, leads to additional global communications which degrade somewhat the parallel performance.

3.4 Parallel Implementation of Cell Computation with Non-Constant Coefficients

$$\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t/2} = \nu(x,y)(\mathbf{v}^*_{xx} + \mathbf{v}^n_{yy}) - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.15}$$

$$\frac{\mathbf{v}^{n+1} - \mathbf{v}^*}{\Delta t/2} = \nu(x,y)(\mathbf{v}^*_{xx} + \mathbf{v}^{n+1}_{yy}) - \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.16}$$

Equation (3.15) is equivalent to Eq. (3.1) and Eq. (3.16) to Eq. (3.2) but the difference lies in the fact that $\nu(x,y)$ is not constant. For example, it can be as the Eq. (4.3) in section 4.

We divide Eqs. (3.15) and (3.16) by $\nu(x,y)$ and denote $\lambda_0^2 \triangleq (\frac{\Delta t}{2})^{-1}$.

Then, Eqs. (3.15) and (3.16) become

$$\lambda_0^2 \nu(x,y)^{-1}(\mathbf{v}^* - \mathbf{v}^n) = \mathbf{v}^*_{xx} + \mathbf{v}^n_{yy} - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.17}$$

$$\lambda_0^2 \nu(x,y)^{-1}(\mathbf{v}^{n+1} - \mathbf{v}^*) = \mathbf{v}^*_{xx} + \mathbf{v}^{n+1}_{yy} - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.18}$$

Then, we have to solve the following pair of equations:

$$\mathbf{v}^*_{xx} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^* = -\mathbf{v}^n_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^n - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.19}$$

which was derived from Eq. (3.17), and

$$\mathbf{v}^{n+1}_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^{n+1} = -\mathbf{v}^*_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^* - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.20}$$

which was derived from Eq. (3.18).

We rewrite Eq. (3.19) as:

$$\mathbf{v}^*_{xx} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^* \triangleq \mathcal{R}_1 \tag{3.21}$$

and Eq. (3.20) as:

$$\mathbf{v}^{n+1}_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^{n+1} \triangleq \mathcal{R}_2 \tag{3.22}$$

where

$$\mathcal{R}_1 = -\mathbf{v}^n_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^n - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.23}$$

and

$$\mathcal{R}_2 = -\mathbf{v}^*_{yy} - \lambda_0^2 \nu(x,y)^{-1}\mathbf{v}^* - \nu(x,y)^{-1} \sum_{q=0}^{J_e-1} \beta_q \left(\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}\right) \tag{3.24}$$

In each processor (domain) we assume that $\bar{\lambda}_0^2 = \lambda_0^2 < \nu >$ where $<>$ denotes the spatial average. Then, we can use the results from section 3.3 and apply them as was done in section 3.2.

4 Model Problem

The algorithm was tested and implemented using the following model problem which is a two-dimensional equation of the Navier-Stokes type [15].

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \nabla^2 \mathbf{v} - \mathbf{N}(\mathbf{v}) + \mathbf{F}(\mathbf{x}, \mathbf{t}), \qquad \text{in } \Omega = (0, 2\pi)^2 \tag{4.1}$$

$$\nabla \cdot \mathbf{v} = 0$$

$$\mathbf{N}(\mathbf{v}) = (\mathbf{v} \cdot \nabla)\mathbf{v} + \frac{1}{2}(\nabla \cdot \mathbf{v})\mathbf{v} \tag{4.2}$$

$$\mathbf{v} = (v_x, v_y), \qquad \mathbf{F} = (F_x, F_y, t), \qquad \mathbf{x} = (x, y)$$

and

$$\nu(x,y) = \frac{1}{1 + \epsilon \cos kx \cos ky} \tag{4.3}$$

where the boundary conditions are

$$\mathbf{v}(x + L_x, y) = \mathbf{v}(x, y + L_y) = \mathbf{v}(x, y) \tag{4.4}$$

and the viscosity for the constant coefficients case is $Re^{-1} = \nu$ and for the non-constant coefficients $\nu = \nu(x, y)$.

The non-linear term is written in the form (4.2) which preserves the energy.

The exact solution $\mathbf{v} = (v_x^{exact}, v_y^{exact})$ is:

$$\begin{aligned} v_x^{exact} &= \cos K_x x \cos K_y y \\ v_y^{exact} &= \cos L_x x \cos L_y y \end{aligned} \tag{4.5}$$

5 Performance Analysis

The 2-D solver was implemented on three different MIMD message-passing multiprocessors: a 60-processor IBM SP2, a 20-processor MOSIX, and a network of 10 Alpha workstations using PVM[8] software package. The same program without any modifications using PVM was running on all multiprocessors. The next section, 5.1, is a description of the three machines.

5.1 The Multiprocessors

The network of Alpha workstations (Alpha farm) used in our study is a distributed-memory farm of a DEC 3000 Model 400/400S AXP system. Each node in the farm is a high-performance desktop system that uses a Digital DECchip 21064 RISC microprocessor running at 133.33 MHz. The processor is based on Digital's Alpha AXP architecture and has about 75Mbytes of memory space available to the user. The Alpha farm consists of 10 computers connected through the Fiber Distributed Data Interface (FDDI). Effective data and message transfers through this interface are realized via the *Giga Switch*, with an average measured data transfer rate of 3-4 Mbytes

Due to memory limitation 75Mbytes is the upper limit for the size of the permitted number of collocation points in each node of the Alpha farm. For the computation of the achieved speedup and efficiency, however, we need measurements of the serial algorithms taken on a number of collocation points N_x greater than what is available on the Alpha farm. In order to get a time estimate for N_x that is bigger than the available memory we extrapolate some results for the serial measurements.

MOSIX is an enhancement of UNIX that allows distributed-memory multicomputers, including a LAN-connected network of workstations (NOW), to share their resources by supporting preemptive process migration and dynamic load balancing among homogeneous subsets of nodes.

The NOW MOSIX is designed to run on configurations that include personal workstations, file servers and CPU servers that are connected by LANs, a shared bus, or fast interconnection networks. In these configurations each workstation (node) is an independent computer, with one or more (shared-memory) processors, local memory, and communication and I/O devices. Users interact with the multicomputer through their own workstations.

Parallel applications can be executed on MOSIX by simply creating many processes, just like a single-machine environment. The operating system automatically attempts to optimize the process allocations and balance the load.

The system image model of MOSIX is a NOW, in which all the user's processes seem to run at the user's "home" workstation. Each new process is created at the same site (or sites) as its parent process. All the processes of each user have the execution environment of the user's home workstation. Processes that migrate to other (remote) workstations use local resources if possible, but interact with the user's environment through the user's workstation.

The NOW MOSIX is compatible with BSDI's BSD/OS Release 2.0, which is based on Release 4.4BSD-Lite from the Computer Systems Research Group at UC Berkeley. The current implementation is for Intel's X86/Pentium-based workstations and PCs.

IBM SP2 has 64 processors, 62 of which are of the type Thin Node 2; each node includes 128 Mbytes memory and a 4.5-Gbyte internal disk. The two remaining processors are of type Thin Node 2, in which each node has 512 Mbytes of memory and 4.5-Gbyte internal disk. Each processor has a POWER2 architecture RS/6000 processor running at 66.7 MHz. The processors are interconnected with a multistage switch made up of bidirectional 4×4 crossbars. The SP2 is theoretically capable of 266 Mflops and it contains the AIX Windows Environment/6000.

Notation:

p	-	number of processors of processors in one direction
n	-	number of points in a subdomain across one side of the cell (total of n^2 points in the domain).
$N_x = np$	-	global number of points in the x direction of the cell
$N_y = np$	-	global number of points in the y direction of the cell
n_t	-	number of time steps
n_R	-	number of Richardson iterations
$2N_\epsilon$	-	number of points in the overlapping region. It appeared in figures 1 and 2
$N_{LCT} = n + 2N_\epsilon$	-	the length of the local Fourier basis (LFB)
$N_{FFT} = 2N_{LCT}$	-	the size of the fft
T_s	-	serial time (for a single processor)
T_p	-	parallel time (for p processors)
$S = T_s/T_p$	-	speedup
$E = S/p$	-	efficiency

5.2 Serial timing

To obtain a realistic performance analysis we need timing results of serial programs with very large number of points. Due to memory limitation there is an upper limit for the number of collocation points per processor. For meaningful comparison of the achieved speedup we have to obtain timing results of a very large serial programs that are the counterpart of large parallel programs. Therefore, we use extrapolation methods to estimate the timing of serial programs that do not fit into a single processor.

To compute and estimate of the parallel speedup of the 2-D solver we have to obtain the timing of the best available serial algorithm running on a single processor using the same global resolution $N_x \times N_y$. The serial algorithm is based on the pseudo-spectral Fourier method in space using global FFT in both directions. Both serial and parallel algorithms use the same time stepping scheme. In table 1 we present the best achievable serial solver for the 2-D Navier-Stokes equations.

$N_x = N_y$	ALPHA	MOSIX	SP2
256	71.8	93.1	20.0
512	269.6	348.2	81.2
1024	1046.5	1514.6	365.5
2048	4374.3	6288.9	1571.5

Table 1: Serial timing (in seconds); $n_t = 4,\ n_R = 5$.

If the local resolution n (where n is the number of points in the subdomain across one side, x or y, of the cell) is fixed and the number of processors p grows, then the resolution $N_x \times N_y$, $N_x = np$, may become too large so that the total memory required for the serial solver is beyond the capacity of a single processor. Nevertheless, we can estimate the serial timing by extrapolating the data in table 1 for large N_x and N_y. In our serial algorithm the most time consuming operation is the FFT. Other operations such as copying of arrays or pointwise arithmetic operations have a computation count proportional to the number of grid nodes.

In order to evaluate the achieved speedup by the various algorithms serial timing has to be measured and analyzed. The dependence of the serial time T_s (table 1) on the number of points is not smooth (even not monotonic). The reason is that we are using Temperton's FFT algorithm which depends how the number N_x (or N_y) is factorized into its primitive multipliers: $N = 2^i\ 3^j\ 5^k$ where i, j, k are integers. In other words, the performance of the algorithm is different for numbers which are powers of $2, 3$ or 5 only or that contain mix of different multipliers. However, if the size of the FFT is based on one factor only, say $N = 2^i$, then the dependence of T_s on N is depicted by a smooth curve. For further estimation of the speedup, the serial timing for an arbitrary N (not necessarily based only on $2, 3$ and 5 separately) is required. We estimate it by interpolation of known values provided the dependence is smooth. Correspondingly, the serial time can be estimated as follows:

$$T_s = \mu_1 N_x N_y (\log N_x + \log N_y) + \mu_2 N_x N_y \tag{5.1}$$

where the first part is the time for performing the FFT, and the second part is proportional to the resolution. Assume that $N_x = N_y = N$ then we can rewrite Eq. 5.1 as

$$T_s = 2\mu_1 N^2 \log N + \mu_2 N^2. \tag{5.2}$$

The parameters μ_1 and μ_2 are evaluated by the least square method. The values for each multiprocessor are presented in table 2 for $N = 1024$.

	ALPHA	MOSIX	SP2
μ_1	$1.7\ e-5$	$7.1\ e-5$	$2.7\ e-5$
μ_2	$8.5\ e-4$	$7.2\ e-4$	$6.8\ e-5$

Table 2: Parameters for the extrapolation equation 5.2 for the serial timing when $N = 1024$.

The coefficients μ_1 and μ_2 (Eq. (5.1)) in the "pure" FFT algorithm are independent of the total number of points $N_x N_y$ in the region. However, because of the presence of other factors in the program and also due to some particular features of the multiprocessors such as the size of the cache and others, this factor depends slightly on N_x and N_y. Since the μ_1 and μ_2 increase monotonically with the numbers N_x and N_y, the expression in Eq. (5.1) and the data in table 1 are used to estimate the minimal serial time through Eq. (5.2).

The plots of the serial timing are shown in Fig. 6. The values to the right of the dashed vertical line are obtained using the extrapolation Eq. (5.2).

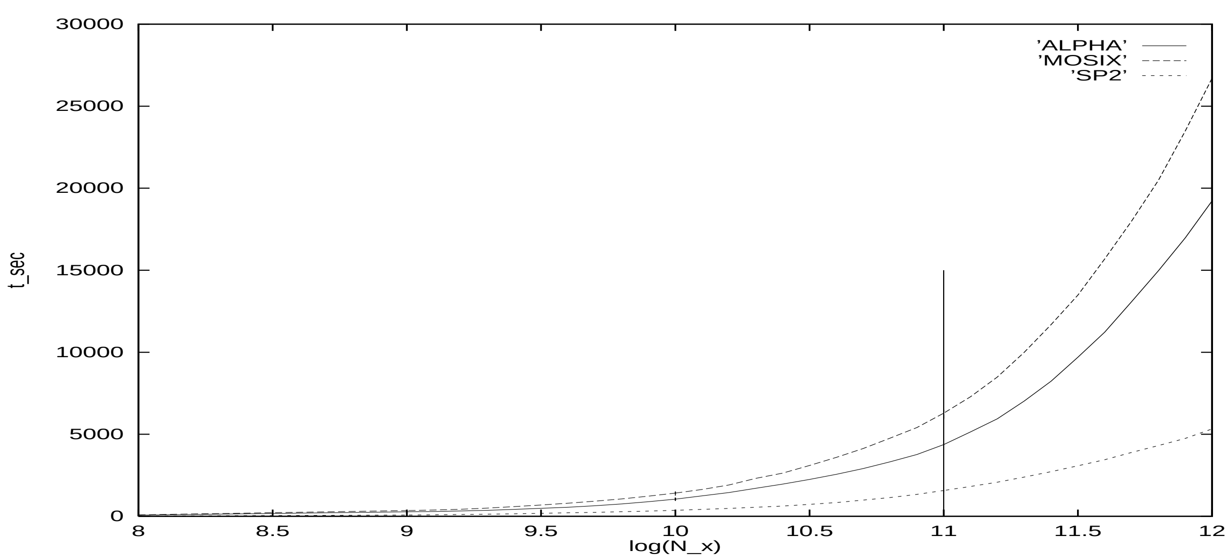

Figure 6: Serial timing on three MIMD computers: $N_x = N_y = 2048$, $n_t = 4$, $n_R = 5$. The values to the right of the vertical line are obtained using the extrapolation Eq. (5.2).

5.3 Parallel performance

5.3.1 Parallel timing

Usually, the FFT is used in the periodic direction and the DCT is applied to (extended and periodized) the local functions across the cell.

Our problem is mapped into a domain which is decomposed into cells. We implemented the LFB technique in the x and y directions separately.

In the first step we take a horizontal row of cells and transform the source function using the LFB in the y-direction. (The procedure includes an ϵ-extension and smoothing across the horizontal interfaces, anti-symmetrical reflection and application of the FFT on the local period).

In the second step we solve the system of one-dimensional problems of type (2.5) for the Fourier coefficients with the local matching on the vertical interfaces and then transform back into the physical space. The resulting particular solution $u^{(I)}$ will be smooth in each horizontal band except on the horizontal interfaces.

To remove discontinuities from the horizontal interfaces we transform the function $u^{(I)}(x, y)$ using the LFB in the x-direction in each vertical column and match Fourier coefficients on these lines. Then, after the application of the inverse transform into the physical space, we get a vertically smooth solution $u^{(II)}$.

Smoothing in the $y-$direction introduces discontinuities on the vertical interfaces because each Fourier coefficient is matched independently. These jumps can be removed by another matching (in the $x-$direction). Then, this is done in each new cycle along x and y, alternatively. This results in a smoother solution. The application to the LFB in the x and y directions is described in section 2.3.

After the completion of the extension and folding of the multidomain local Fourier (MDLF) method (see section 2.3, [1]-[6]) we apply the Fourier transform to even functions in the subdomains. We can utilize the fact that the the folding procedure, as was explained in section 2.3, is based on the symmetry of the bell functions. Thus, the discrete cosine transform (DCT) should be used rather than the FFT (as was done in [1]). In the 2-D case the time that the FFT and DCT use constitutes about 50% of the total parallel time on ALPHA farm and 40% on MOSIX and SP2 machines. From now on, the DCT is used instead of using the FFT in the pseudospectral phase of the algorithm. Therefore, in table 3 the total parallel time of the algorithm in each cell is measured with the assumption that the kernel of the pseudospectral computation, which is applied on each direction, is based on the DCT.

We used $N_\epsilon = 16$ where N_ϵ as appeared in figures 2 and 1. Thus, the parallel time for the 2-D problem is given in table 3

	$n = 480$			$n = 992$
p	ALPHA	MOSIX	SP2	SP2
2×2	479.8	645.8	131.7	617.7
3×3	497.0	662.7	133.3	618.9
3×4		678.4	141.9	621.1
4×4		693.8	138.0	622.2
4×5		698.3	139.8	616.2
6×6			140.4	624.3
7×7			144.9	613.6
8×8			142.1	625.5

Table 3: Parallel timing (in seconds) on three MIMD machines using DCT: $n_t = 4$, $n_R = 5$, $N_\epsilon = 16$.

We can see from table 3 that the parallel time of the solver stays almost the same as the number of the processors (and the size of the problem) increases. This is because the same work has to be done in the x and y directions separately. The cell topology dictates massive work in both directions which reduces the efficiency of the algorithm. If the algorithm was fully efficient then we expect the parallel time to increase with the increase in the number of the processors. But as we will see in section 5.3.3 we get an average of 0.62 in the efficiency.

5.3.2 Communication timing

In the following we examine the communication aspects of the parallel solver.

The communication characteristics of the three multiprocessors is given in table 4.

	ALPHA	MOSIX	SP2
Communicate rate (Mbit/sec)	24	100	280
Latency (μsec)	100	250	40

Table 4: Communication rates of the multiprocessors

The present algorithm (ADI in time and Richardson iterations for solving ODEs with variable coefficients in space) requires only local communication between the processors (as explained in sections 2 and 3). In table 5 the communication timing is given for $N_\epsilon = 16$, $n_t = 4$, $n_R = 5$. The communication is performed 80 times. The amount of transferred data is 21 MB (independent of the resolution n in subdomains).

p	ALPHA	MOSIX	SP2
2x2	3.5	4.2	3.5
3x3	3.8	4.7	3.6
3x4		4.9	4.4
4x4		5.4	4.6
4x5		5.8	5.1
6x6			4.9
7x7			5.3
8x8			5.2

Table 5: Communication time (in sec.). It is independent of n

As indicated in sections 2 and 3 the parallel solver required local communication between neighboring processors. When we compare the results between tables 3 and 5 we note that the communication time constitutes a small part (some several percents) of the total performance time. The communication time depends only slightly on the number of connected processors. These results indicate that the communication overhead related to communication time is negligible. This is due to to the MDLF methodology as was explained before.

5.3.3 Speedup and efficiency

Speedup and efficiency of the 2-D Navier-Stokes parallel solver are presented in table 6.

	$n = 480$			$n = 992$
p	ALPHA	MOSIX	SP2	SP2
2×2	2.12 (0.53)	2.27 (0.57)	2.52 (0.63)	2.30 (0.58)
3×3	4.66 (0.52)	4.97 (0.55)	5.71 (0.64)	5.44 (0.60)
4×4		8.92 (0.55)	10.52 (0.66)	9.95 (0.62)
6×6			24.06 (0.67)	23.20 (0.66)
7×7			34.35 (0.70)	33.32 (0.68)
8×8			46.06 (0.72)	45.44 (0.71)

Table 6: Speedup and efficiency (in brackets) of the 2-D parallel solver of the Navier-Stokes equations using ADI

From table 6 we can see that the ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem is the main feature of a *scalable* parallel systems. The *scalability* of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to utilize increasing processing resources effectively. We demonstrated that the new proposed parallel solver for the 2-D Navier-Stokes equations using ADI discretization is classified to be called scalable in the above sense as we examined table 6. When the number of processors increases and, therefore, the size of the problem also increases, the parallel efficiency stays almost the same, but the serial computation time increases in accordance with Eq. (5.1). But the average achieved efficiency is 0.61 because of the necessary matching in both directions.

From these results we see that the scalability function is approximately constant for all considering computers. It can be explained by realizing first that the communication time is a small part of the total computation time, and secondly, the time spent for synchronization of the different parts of our computational scheme was not too large. Hence, we can deduce that the 2-D ADI parallel algorithm is scalable. The efficiency of ADI algorithm is lower than of 2-D and 3-D Navier-Stokes schemes[6]. This is because in the computation in the cell topology we need to apply the local Fourier basis in both directions.

6 Matching

In the following we see how the local matching on the interfaces is performed. It is based on a specific instance of the model problem in section 4. These examples demonstrate how the matching on the domain interfaces generate a complete solution which is composed from different pieces. We use the following parameters:

Number of processors	$4 = 2 \times 2$
Viscosity coefficient	$\nu = 0.001$
Time step	$\Delta T = 0.002$
Size of problem	128×128 (with 4 processors)
number of overlapping points	32

In the exact solution of (4.5) we use different wavenumber. We assume that the periodic boundary conditions.

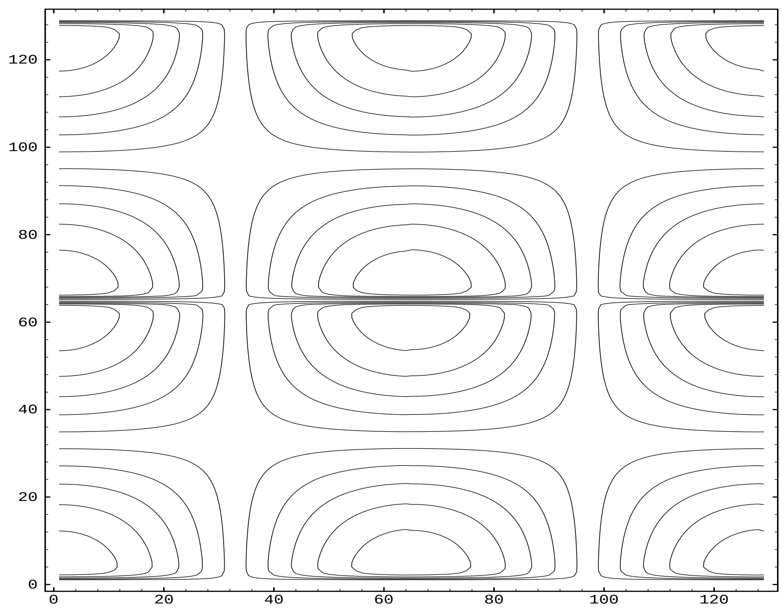

Figure 7: Before matching in the x direction, where the wavenumber are $K_x = K_y = L_x = L_y = 1$

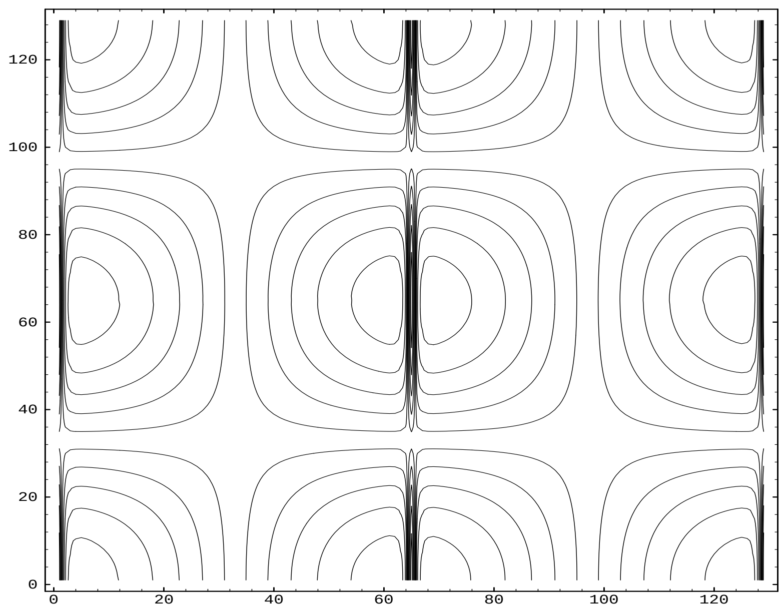

Figure 8: Before matching in the y direction, where the wavenumber are $K_x = K_y = L_x = L_y = 1$

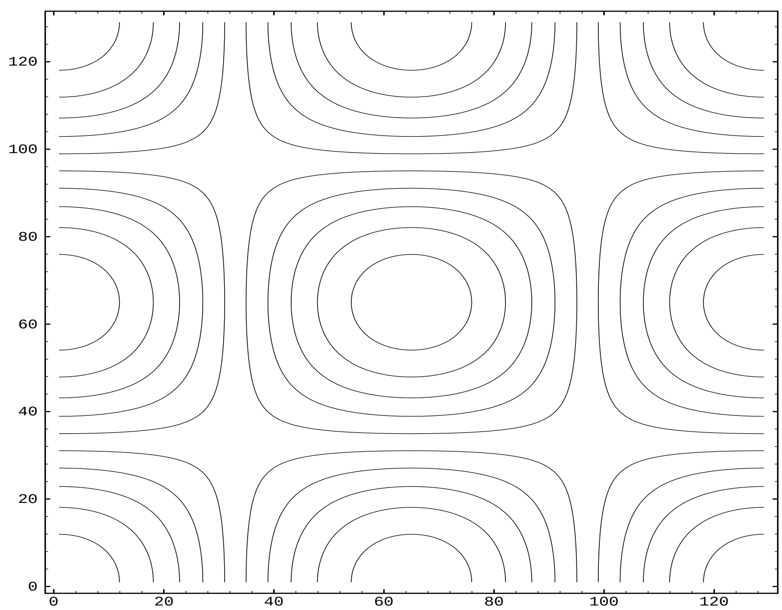

Figure 9: After matching where the wavenumber are $K_x = K_y = L_x = L_y = 1$

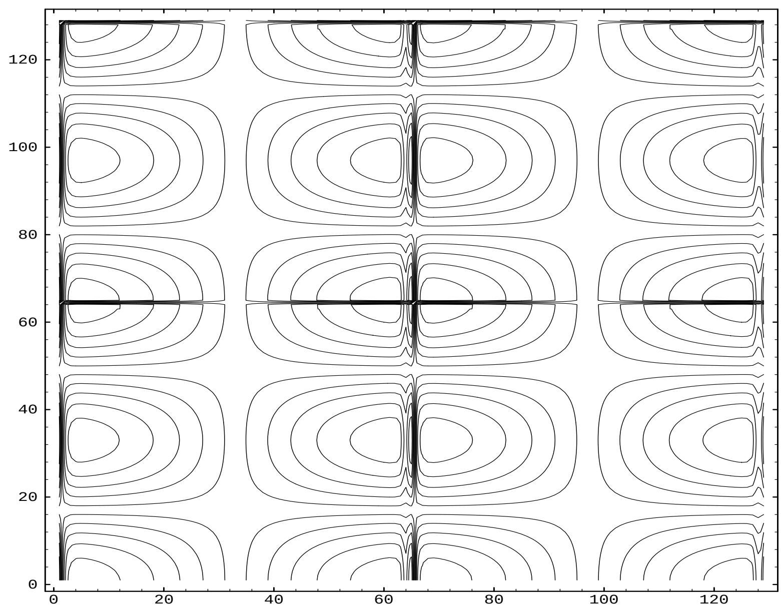

Figure 10: Before matching in the x and y directions where the wavenumber are $K_x = 2, K_y = L_x = 1, L_y = 2$.

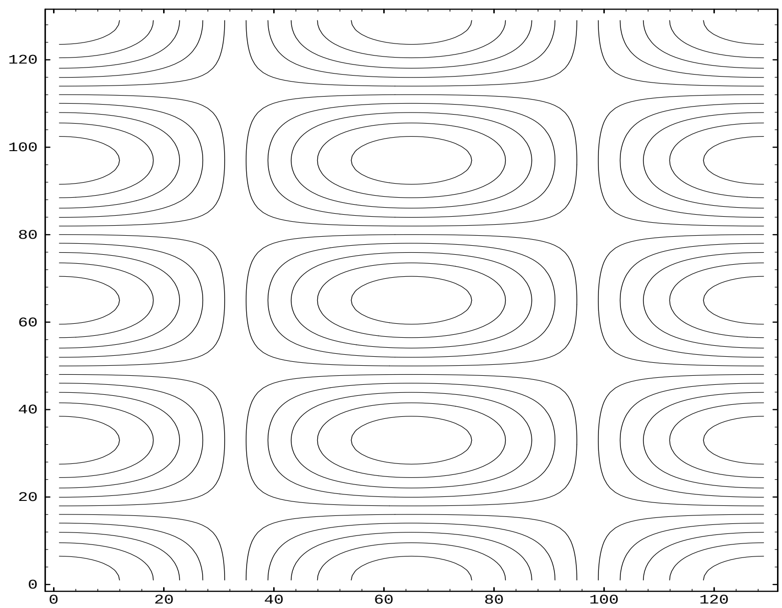

Figure 11: After matching where the wavenumber are $K_x = 2, K_y = L_x = 1, L_y = 2$.

Conclusions

The paper proposed and implemented a new algorithm for the parallel solution of two-dimensional Navier-Stokes type equation with constant and non-constant coefficients which is mapped onto cell topology. The application to problems with non-constant coefficients is done via spectral multidomain preconditioner. This approach is efficient when we have good local approximations in each subdomain (small variations in the coefficients from their mean values). This is guarantee when the computational domain is divided into a large enough number of subdomains. The main achievement here is that we are able to handle decomposition of the domain into cells that is the decomposed in both directions. The ADI scheme is used for the reduction of a 2-D problem to a collection of uncoupled 1-D ODE's. The algorithm was implemented on three different message-passing MIMD multiprocessors using the same PVM program.

The proposed algorithm has the potential for treating efficiently complex geometries. This is a work in progress.

References

[1] A. Averbuch, M. Israeli, L. Vozovoi, Parallel Implementation of Non-Linear Evolution Problems using Parabolic Domain Decomposition, *Parallel Computing*, **21**, No. 7(1995), pp. 1151-1183.

[2] A. Averbuch, M. Israeli, L. Vozovoi, Spectral Multidomain Technique with Local Fourier Basis, *J. of Scientific Computing*, Vol. 8, No. 2, pp. 135-149, June 1993.

[3] M. Israeli, L. Vozovoi, A. Averbuch, Parallelizing Implicit Algorithms for Evolution Equations by Parabolic Domain Decomposition, *J. of Scientific Computing*, Vol. 8, No. 2, pp. 151-166, June 1993

[4] M. Israeli, L. Vozovoi, A. Averbuch, Domain decomposition methods for solving parabolic PDEs on multiprocessors, *Applied Numerical Mathematics*, **12** (1993), pp. 193-212.

[5] L. Vozovoi, M. Israeli, A. Averbuch, Spectral Multidomain Technique with Local Fourier Basis II: Decomposition into Cells, *J. of Scientific Computing*, Vol. 9, No. 3, pp. 311-326, 1994.

[6] A. Averbuch, L. Ioffe, M. Israeli, L. Vozovoi, Highly Scalable two- and Three-Dimensional Navier-Stokes Parallel Solvers on MIMD Multiprocessors, J. of Supercomputing, Vol. 11, No. 1, pp. 7-39, 1997.

[7] R. Coifman, Y. Meyer, Remarques sur l'analyse de Fourier à fenêtre, série, *C.R.Acad.Sci.* Paris **312** (1991), p. 259-261.

[8] A. Geist, A. Beguelin et. al., PVM: parallel virtual machine, *The MIT Press*, Cambridge, (1994).

[9] A. Barak, S. Guday, R. Wheeler, The MOSIX Distributed Operating System, Load Balancing for UNIX. *Lecture Notes in Computer Science*, Vol. 672, Springer-Verlag, 1993.

[10] A. Barak, O.Laden, Z. Yarom, The NOW MOSIX and its Preemptive Process Migration Scheme, *IEEE TCOS*, Vol. 7, No. 2, pp. 5-11, 1995.

[11] M. Israeli, A Fast implicit numerical method for time dependent viscous flows, *Studies in Applied Mathematics*, **49**, N 4, pp. 327-349, (1970)

[12] G.E. Karniadakis, M. Israeli, S. Orszag, High-order splitting methods for the incompressible Navier-Stokes equations, *J.Comp.Phys.*, **97**, No. 2, (1991), p. 414-443

[13] S. A. Orszag, Spectral methods for problems in complex geometries, *Numerical methods for PDE's*, 1979.

[14] C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, Spectral methods in fluid dynamics, *Series in Computational Physics*, Springer-Verlag, 1989.

[15] R.Temam, Stability Analysis of the Nonlinear Galerkin Method, Math. of Computation **57**, No. 196, (1991), p. 477-505.

[16] L. Vozovoi, M. Israeli, A. Averbuch, Analysis and application of Fourier-Gegenbauer method to stiff differential equations, *SIAM J. Numerical Analysis*, **33**, No. 5, pp. 1844-1863, October 1996.